Master's Thesis

An Expectation Maximization Method to Learn the Group Structure of Deep Neural Network

Subin Yi

Department of Computer Science and Engineering

Graduate School of UNIST

2018

An Expectation Maximization Method to Learn the Group Structure of Deep Neural Network

Subin Yi

Department of Computer Science and Engineering

Graduate School of UNIST

An Expectation Maximization Method to Learn the Group Structure of Deep Neural Network

A thesis
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Subin Yi

12/15/2017
Approved by
Jaesik Choi
Advisor
Jaesik Choi

An Expectation Maximization Method to Learn the Group Structure of Deep Neural Network

Subin Yi

This certifies that the thesis of Subin Yi is approved.

12/15/2017

Advisor: Jaesik Choi

Jun Moon

Sung Ju Hwang

Abstract

Analyzing multivariate time series data is important for many applications such as automated control, sensor fault diagnosis and financial data analysis. One of the key challenges is to learn latent features automatically from dynamically changing multivariate input. Convolutional neural networks (CNNs) have been successful to learn generalized feature extractors with shared parameters over the spatial domain in visual recognition tasks. For high-dimensional multivariate time series, designing an appropriate CNN model structure is challenging because the kernels may need to be extended through the full dimension of the input volume. To address this issue, we propose an Expectation Maximization (EM) method to learn the group structure of deep neural networks so that we can process the multiple high-dimensional kernels efficiently. This algorithm groups the kernels for each channel using the EM method and partition the kernel matrix into a block matrix. The EM method assumes the Gaussian Mixture Model (GMM) and the parameters of the GMM is updated together with the parameters of deep neural network by end-to-end backpropagation learning.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN artificial neural network. 1, 4, 9, 18

CNN convolutional neural network. 1–4, 9, 13, 14, 18

DNN deep neural network. 9

EM expectation maximization. 1, 2, 6, 7, 10, 13, 18

GMM Gaussian mixture model. 1–3, 5, 7, 18

HAR human activity recognition. 9

LASSO least absolute shrinkage and selection operator. 9

LSTM long short-term memory. 1, 9

MAP maximum a posteriori. 6

ML maximum likelihood. 6

MLP multilayer perceptron. 18

RCNN recurrent convolutional neural network. 4, 9

RMSE root mean squared error. 3, 11, 15

RNN recurrent neural network. 1, 9

Chapter I

Introduction

Multivariate time series data analysis has broad application in many domains such as system control, financial analysis, and so on. So far, nonparametric Bayesian models [8, 17] and model based control methods [14] have been popular tools for analyzing multiple time series. However, these approaches are highly sensitive to the model parameters meanwhile finding the proper parameters for high-dimensional data is a painful work. Moreover, new and complicated fields, such as bioinformatics, that require more accurate analytic methods are emerging with the development of the technology while the existing applications are also becoming more complex and exacting. Accordingly, more accurate and sophisticated time series analysis techniques became necessary.

Machine learning methods have been successful in many domains that have been stagnant such as digit recognition, language translation, go playing and so on. Therefore, there has been researches which adopts artificial neural networks (ANNs) to handle multivariate time series data. Autoencoders [2, 23] train model parameters in an unsupervised manner by specifying the output value same as the input. Recurrent neural network (RNN) [19] and long short-term memory (LSTM) [6] find temporal features from the multivariate sequences using the recurrent transition function between time steps. Convolutional neural network (CNN) can be also used as feature extractors by computing 1D convolutions over fractions of multiple time series. Most of existing neural network models assume (shared) fully connected networks under the Markov assumption when they deal with temporal data. Thus, such models are often not precise enough to deal with high-dimensional multivariate datasets, unfortunately.

To process high-dimensional data efficiently, we train the group structure of deep convolutional neural networks using expectation maximization (EM) [1, 15] method. This method uses the mixture model clustering, specifically the Gaussian mixture model (GMM), to group the nodes based on the distributions of input node's connections to the next layer then the EM algorithm is used to optimize the mixture model parameters. After grouping, each group's mean values are compared and weaker connections are eliminated to turn the weight matrix into a block matrix as in the Figure 1.

The parameters for the GMM are updated together with the neural network parameters by backpropagation algorithm.By adding the likelihood function of EM algorithm to the cost function, the model can be optimized to maximize the GMM likelihood function while minimizing the network's cost function.

This EM method learns the group structure of the connections between the network layers by end-to-end learning algorithm and disconnects the weights that are not from the group that has the strongest connection to each output node. This process reduces the redundant kernels of existing CNNs and makes it possible to build a lighter model that performs as good as the model with more parameters.

In Chapter II, we will briefly explain some of the background knowledge for our proposed method followed by the Chapter III, related works. Then in the Chapter IV we will introduce the EM method for learning group structure and explain the detailed algorithm. In Chapter V, we show that adding grouped convolutional layers can accelerate the training by exploiting the larger number of parameters at the early step of learning and pruning the network afterwards from the experiments with MNIST hand-written digit dataset.

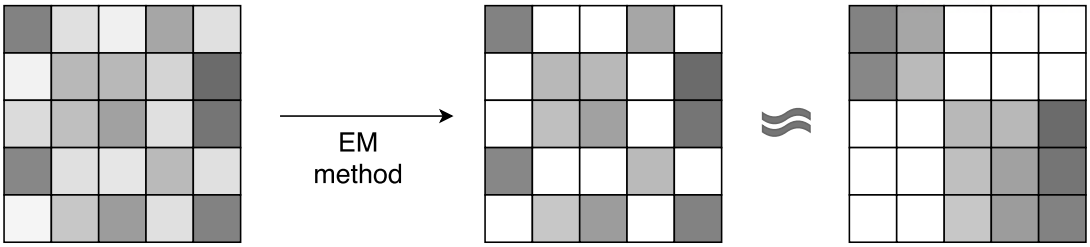


Figure 1: Visualization of how the proposed EM method works. It learns the group structure of a network using GMM to replace a weight connection with a simpler block matrix. It first groups the input nodes based on their connections to the output layer and let the output nodes to choose group that has the strongest connection to them and disconnects the connection with other groups. Organizing the order for the purpose of the visualization, the resulting matrix can be described as a block matrix.

Chapter II

Background

2.1 Convolutional Neural Network (CNN)

A convolutional neural network (CNN) is a multi-layer artificial neural network inspired by the animal visual cortex to solve vision problems. The cortical neurons of visual cortex respond to stimuli from a restricted region of the visual field which is called as a receptive field and the receptive fields of different neurons partially overlaps such that they cover the whole visual field. CNN adopts this idea to build a layer, called a convolutional layer, which receives inputs from a small region of the previous layer using convolution operation.

CNNs are generally composed of a stack of three types of multiple layers: *convolutional layer*, *sub-sampling layer*, and *fully-connected layer*. First, convolutional layers compute convolutions over the input layer. Generally, the input to a convolutional layer is a $w \times h \times d$ sized vector where w is the width, h is the height, and d is depth (or channel), e.g. an RGB image has $w \times h$ pixels and $d = 3$ for RGB channels. The convolutional layer will have f number of filters (or kernels) of a receptive field size $w' \times h' \times d'$ where $w' \leq w, h' \leq h$ with $d' = d$ in most cases but could be less than d. Then starting from the corner of the input, the convolution layer performs dot product of the small piece of the input with the kernel W to produce a single node of one output channel. Then the kernel W is slid along the input with the predetermined stride to produce one channel. Given an input $\mathbf{x}$, the i-th channel of the output layer, $\mathbf{h}^i$ is computed from:

$$\mathbf{h}^i = \sigma(\mathbf{x} \otimes W^i + b^i) \tag{2.1.1}$$

where W^i is the i-th filter, b^i is the i-th bias, $\sigma(\cdot)$ is a nonlinear function such as $sigm$, and $\otimes$ represents the convolution operation described above. Each kernel of the convolutional layer filters out an unique feature from the input layer and produces a feature map. The convolution layer repeats this process for all the f filters to produce f feature maps.

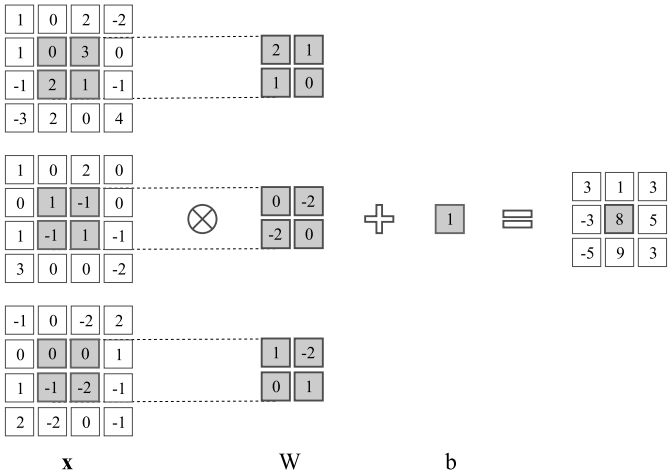


Figure 2: Convolutional layer.

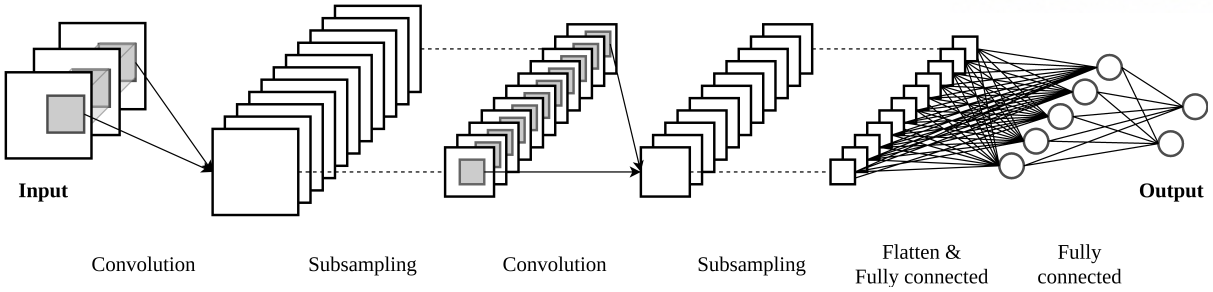


Figure 3: Convolutional neural network example.

Second, the nonlinear sub-sampling layer takes samples from small regions of the input layer using nonlinear functions reducing the size of the layer. The most common choice for this local down-sampling is max-pooling but mean or other methods can also be used. The idea of local sub-sampling is that once a feature is detected, its absolute location is not as important as its relative location. By reducing the dimensionality, it makes the network less sensitive to the locality and lowers the computational complexity [10, 11].

The last type of CNN layers is the fully-connected layer. After convolutional layers and sub-sampling layers produce feature maps, fully-connected layers implement the reasoning using the extracted features to produce the actual output.

Conventionally, CNN consists of alternate layers of convolutional layers and sub-sampling layers on the bottom and several fully-connected layers following them as in the example of the Figure 3. The model receives an input data composed of three channels like general image files and extract the local features via two sets of convolution and sub-sampling layers. Then it processes the extracted features to produce the actual output.

CNNs are trained via error backpropagation method like other ANNs. First a cost function $J(\theta|\mathbf{x}, \mathbf{t})$ to minimize is defined where θ is the set of parameters and $\mathbf{t}$ is the target output. The parameters on the last layer are updated using gradient descent method minimizing the cost function and then the error is propagated backward to update the previous layer's parameters until it reaches the first layer and updates all the parameters in the network.

The local computations of CNNs not only reduce the memory burden but also improve the classification performance. They have been very successful in visual processing problems such as classification [11, 9], object detection, and semantic segmentation [18, 13]. Furthermore, it has been applied to natural language processing [7, 3] and time series data processing. Also, a recurrent variant of CNN, named as recurrent convolutional neural network (RCNN) [5] showed state-of-the-art performance in object recognition problem [12].

2.2 Mixture Models

2.2.1 Mixture Models

A mixture model is a probabilistic model which represents the probability of the observations in the overall population by mixture of distribution. In many real life examples, the data to be modeled does not follow one probabilistic model such as uniform or Gaussian. A mixture model combines multiple models to better represent the multimodal case in which the data is composed of observations from more

than one models. Using K number of probabilistic models, the probability distribution of an observation **x** is described as below:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k p_k(\mathbf{x}|\boldsymbol{\theta}) \tag{2.2.1}$$

where $\boldsymbol{\theta}$ is the parameters of the base distributions. This is the weighted sum of the base distributions p_k's and the variables π_k's are the mixing weights or the mixing coefficients of this mixture model. They are based on the information about the base distributions which can never be explicitly observed. Hence, it is formulated with respect to a latent variable $\mathbf{z} = [z_1...z_K]$. $\mathbf{z}$ is the vector of K binary random variables having a 1-of-K representation so that $z_k \in \{0, 1\}, \forall k \in \{1, ..., K\}$, and $\sum_k z_k = 1$. Then the mixing coefficients π_k's can be formally defined as below:

$$\pi_k = p(z_k = 1) \tag{2.2.2}$$

satisfying $\sum_{k_1}^{K} \pi_k = 1$ together with $0 \leq \pi_k \leq 1$. Then the Eq. 2.2.1 tells us how the observation **x** was generated from the perspective of the mixture model. First, the distribution k was chosen with probabilities given by the mixing coefficients and then one observation was generated according to the distribution.

Another way to view the mixture models is to consider each component distribution as one cluster and to separate the data points according to the clusters. From this perspective of view, what the variable **z** represents is just the cluster label. To accomplish the clustering, the posterior distribution $p(z_k = 1|\mathbf{x}_i)$ is used, which is the likelihood of $\mathbf{x}_i$ belonging to k-th cluster given the data point. This is called the *responsibility* of k-th cluster for the i-th data point and can be inferred using Bayes' Rule as below:

$$\begin{aligned} \gamma_{ik} &= p(z_k = 1|\mathbf{x}_i, \boldsymbol{\theta}) \\ &= \frac{p(z_k = 1|\boldsymbol{\theta})p(\mathbf{x}_i|z_k = 1, \boldsymbol{\theta})}{\sum_{j=1}^{K} p(z_j = 1|\boldsymbol{\theta})p(\mathbf{x}_i|z_j = 1, \boldsymbol{\theta})} \end{aligned} \tag{2.2.3}$$

This procedure is called *soft clustering* as it gives the probability of the data point belonging each cluster.

2.2.2 Gaussian Mixture Model

GMM is one example of the mixture models whose base distributions are Gaussian distributions. It can be written as a superposition of K base Gaussian distributions with the mean and covariance matrix of the k-th component as $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$. Formally, the distribution $p(\mathbf{x}|\boldsymbol{\theta})$ is defined as below when $\mathbf{x}, \boldsymbol{\mu}$ are L-dimensional vectors and $\boldsymbol{\Sigma}$ is LxL matrix:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{2.2.4}$$

$$= \sum_{k=1}^{K} \pi_k \frac{1}{\sqrt{(2\pi)^L |\boldsymbol{\Sigma}_k|}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)). \tag{2.2.5}$$

Here, $\boldsymbol{\theta}$ is a set of the base normal distributions' parameters, $\{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, ..., \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K\}$.

2.3 EM Algorithm

Maximum likelihood (ML) and maximum a posteriori (MAP) are the popular ways to optimize models in statistics and machine learning. Computing ML or MAP parameter estimate is relatively easy given the complete data, $\mathcal{D}$. However, when some of the variables are missing or latent, computing the estimates becomes hard. EM algorithm is an iterative algorithm for computing the ML/MAP estimates given data with latent variables first introduced in [4]. For a complete data D, let $\mathbf{x}_1, ..., \mathbf{x}_N$ be the observed variables and $\mathbf{z}_1, ..., \mathbf{z}_K$ be other latent variables where $N >> K$. The goal of the EM algorithm is to maximize the log likelihood of the observed data defined as below where $\boldsymbol{\theta}$ represents the parameters to estimate.

$$\ell(\boldsymbol{\theta}) = \log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^{N} \log \Big[\sum_{\mathbf{z}_i} p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) \Big] \tag{2.3.1}$$

Unfortunately, this is hard to compute due to the inner summation. Instead, EM defines the complete data log likelihood, $\ell_c(\boldsymbol{\theta})$, as below:

$$\ell_c(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}), \tag{2.3.2}$$

Because the complete data log likelihood is not applicable due to the latent variables, EM maximizes the expected log likelihood under the posterior distribution of the latent variables, $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$, defined as :

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) = \mathbb{E}[\ell_c(\boldsymbol{\theta})|\mathcal{D}, \boldsymbol{\theta}^{t-1}] \tag{2.3.3}$$

where t is the current iteration index.

EM algorithm is therefore composed of two steps: **E-step** and **M-step**. **E-step** is to compute $Q(\boldsymbol{\theta}', \boldsymbol{\theta})$ and **M-step** is to update the parameters $\boldsymbol{\theta}$ by solving the optimization problem given below:

$$\boldsymbol{\theta}^t = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) \tag{2.3.4}$$

2.4 EM for GMMs

2.4.1 Expected Log Likelihood Function

The EM algorithm for GMM maximizes $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$ given as:

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) &= \mathbb{E}\Big[\sum_{i=1}^{N} \log p(\mathbf{z}_i, \mathbf{x}_i|\boldsymbol{\theta})\Big] \\ &= \sum_{i=1}^{N} \mathbb{E}\Big[\log\Big[\prod_{k=1}^{K}\{\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k)\}^{z_{ik}}\Big]\Big] \\ &= \sum_{i=1}^{N}\sum_{k=1}^{K} \mathbb{E}[z_{ik}] \log(\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k)) \\ &= \sum_{i=1}^{N}\sum_{k=1}^{K} p(z_{ik}|\mathbf{x}_i, \boldsymbol{\theta}^{t-1}) \log(\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k)) \\ &= \sum_{i=1}^{N}\sum_{k=1}^{K} \gamma_{ik} \log \pi_k + \sum_{i=1}^{N}\sum_{k=1}^{K} \gamma_{ik} \log p(\mathbf{x}_i|\boldsymbol{\theta}_k) \end{aligned} \tag{2.4.1}$$

where $\mathbf{z}_i$ is same as the vector $\mathbf{z}$ introduced above corresponding to the variable $\mathbf{x}_i$ and z_{ik} is the k-th element of $\mathbf{z}_i$. The variable $\gamma_{ik} = p(z_{ik} = 1|\mathbf{x}_i, \boldsymbol{\theta}^{t-1})$ is the *responsibility* that the k-th Gaussian distribution takes for the observed data $\mathbf{x}_i$.

2.4.2 E step

In the **E step** of the EM algorithm, γ_{ik} value is updated following the equation below:

$$\begin{aligned} \gamma_{ik} &= p(z_{ik} = 1|\mathbf{x}_i, \boldsymbol{\theta}^{t-1}) \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k^{t-1}, \boldsymbol{\Sigma}_k^{t-1})}{\sum_{j=1}^{k} \pi_j \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_j^{t-1}, \boldsymbol{\Sigma}_j^{t-1})} \end{aligned} \tag{2.4.2}$$

The γ_{ik} is the *responsibility* that the k-th Gaussian distribution takes for the observed data $\mathbf{x}_i$. The variable $\gamma(z_k) = p(z_k = 1|\mathbf{x}, \boldsymbol{\theta}^{t-1})$ can be considered as the posterior probability corresponding to π_k once $\mathbf{x}$ is observed.

2.4.3 M step

In the **M step**, the parameters are updated toward maximizing the Q function w.r.t. $\boldsymbol{\pi}$, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$. To maximize the expected log likelihood function, set the derivative of the equation 2.2.5 with respect to the means $\boldsymbol{\mu}_k$ to be 0. Then we get

$$-\sum_{i=1}^{N} \frac{\pi_k p(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\boldsymbol{\Sigma}_j \pi_j p(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \boldsymbol{\Sigma}_k(\mathbf{x}_i - \boldsymbol{\mu}_k) = 0. \tag{2.4.3}$$

Multiplying by $\mathbf{\Sigma}_k^{-1}$ and rearranging, we get

$$\mu_k = \frac{1}{\gamma_k} \sum_{i=1}^{N} \gamma_{ik} \mathbf{x}_i \tag{2.4.4}$$

where $\gamma_k = \sum_{i=1}^{N} \gamma_{ik}$. This γ_k can be interpreted as the effective number of points assigned to cluster k. Similarly, if we take derivate with respect to Σ_k and π_k respectively and follow the same reasoning, we get

$$\Sigma_k = \frac{1}{\gamma_k} \sum_{i=1}^{N} \gamma_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \tag{2.4.5}$$

$$\pi_k = \frac{\gamma_k}{N} \tag{2.4.6}$$

After computing the new estimates, the parameters are updated for $k = 1, .., K$ and the updates are iterated until the log likelihood estimate converges as in the Algorithm 1.

Algorithm 1 EM Algorithm for GMM

1: **procedure** EMforGMM($\mathbf{X}$, N, K)
2: $t := 0$.
3: Initialize the means $\boldsymbol{\mu}_k^t$, covariance matrices $\mathbf{\Sigma}_k^t$, and mixing coefficients π_k^t.
4: Evaluate initial Q value.
5: **while** The convergence criterion is not satisfied **do**
6: $t := t + 1$.
7: Evaluate the responsibilities, γ_{ik} by

$$\gamma_{ik}^t = \frac{\pi_k^{t-1} \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k^{t-1}, \mathbf{\Sigma}_k^{t-1})}{\sum_{j=1}^{k} \pi_j^{t-1} \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_j^{t-1}, \mathbf{\Sigma}_j^{t-1})}.$$

8: Update parameters using the current γ_{ik}^t as

$$\mu_k^t = \frac{1}{\gamma_k^t} \sum_{i=1}^{N} \gamma_{ik}^t \mathbf{x}_i$$

$$\Sigma_k^t = \frac{1}{\gamma_k^t} \sum_{i=1}^{N} \gamma_{ik}^t (\mathbf{x}_i - \boldsymbol{\mu}_k^t)(\mathbf{x}_i - \boldsymbol{\mu}_k^t)^T$$

$$\pi_k^t = \frac{\gamma_k^t}{N}$$

9: Check for the convergence of either the log likelihood or the parameter values.

Chapter III

Related Works

RNN and LSTM are the popular choices when ANNs are used to solve temporal problems, however, recently there have been several works that used CNN and its variant models such as RCNN to deal with the time series data.

Yang et al. [21] used CNN for human activity recognition (HAR) problem, in which inputs are multiple bodyworn inertial sensor signals of specific length and outputs are predefined activities. They showed that the deep convolutional neural networks can be used as an automatic feature extractor for the multichannel time series.

Ordóñez and Roggen [16] introduced a deep neural network (DNN) referred to as DeepConvLSTM which combined CNN and LSTM for wearable activity recognition. The difference of the DeepConvLSTM model and the CNN of Yang et al. [21] is the layers that follows the convolutional layers. DeepConvLSTM used LSTM blocks instead of fully-connected layers so that the convolutional layers act as feature extractors and the following LSTM layers model the temporal dynamics of the extracted features.

Zheng et al. [24] proposed multi-channels deep convolutional neural networks (MC-DCNN) for multivariate time series classification. They separate multivariate time series into multiple univariate ones and stacked individual convolutional and sub-sampling layers for each time series. The last activations of the individual sub-models were concatenated and processed by fully-connected layers. Their model is similar with our model if we set the number of clusters to be same as the number of time series. However, in MCDCNN does not let the information sharing at the convolutional layer level while our method allows information from the variables in the same group to be mixed. Also, the group can change at every layer and the group structure can be trained by end-to-end learning.

There has been also many researches about regularizing the ANNs. One popular way is to train the parameters by minimizing a regularized cost function. Inspired by the least absolute shrinkage and selection operator (LASSO) algorithm [20], the Lasso regularization performs both the regularization and variable selection by bounding the l_1 norm of the weights.

Yuan and Lin [22] considered the case that the variables are structured into K groups and proposed group lasso method. When the variables can be divided into K different groups and the group memberships are given, only part of the groups are selected and contribute to the output. When the size of the groups are 1, it works same as the lasso regularization.

Chapter IV

EM Method to Learn the Group Structure of Deep Neural Networks

4.1 EM Method

Consider a layer of a neural network which receives the input variables, $X = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and produces the output $\mathbf{H} = \{\mathbf{h}_1, ..., \mathbf{h}_M\}$. Let $W \in \mathbb{R}^{N \times M}$ be its weight matrix and $\mathbf{b} = b_1, ..., b_M$ be the bias. Then the output layer $\mathbf{H}$ can be formally defined as below:

$$\mathbf{H} = \sigma(X * W + b)$$

$\sigma(\cdot)$ is a nonlinear function and $*$ is matrix multiplication. Also, each element h_i in $\mathbf{H}$ can be written as :

$$\mathbf{h}_i = \sigma(\sum_{j=1}^{N} \mathbf{x}_j w_{j,i} + b_i)$$

where $w_{j,i}$ is the j-th row, i-th column element in W.

The goal is to restrict the weight connection between layers so that the weight matrix be a block matrix form as in the Figure 1 by learning the group structure of a network. To achieve the goal, we first group the rows of the matrix (Figure 4). Each row of the matrix W represents each input variable $\mathbf{x}_i$'s connections to the output layer $\mathbf{H}$.

Formally, we substitute $\mathbf{w}_i = [w_{i,1} \cdots w_{i,M}]$ for the $\mathbf{x}_i$ in the Section 2.4 and calculate the expected likelihood function $Q(\theta, \theta^{t-1})$ in Equation 2.4.1. Then $\mathbf{w}_i$ are grouped into K clusters in a way that $\mathbf{w}_i$ falls into the k-th cluster such that the responsibility γ_{ik} has the greatest value among the $k \in 1, ..., K$. Then, the mean vectors $\boldsymbol{\mu}_k$ of the Gaussian distributions are compared with each other to find out in which cluster k $\mu_{k,j}$, the j-th element in $\boldsymbol{\mu}_k$, has the greatest value. For the j-th column, the row i that falls into the k-th cluster where $\mu_{k,j}$ has the largest value remains and other rows are erased to 0 (Figure 4) so that only the strong connections remain.

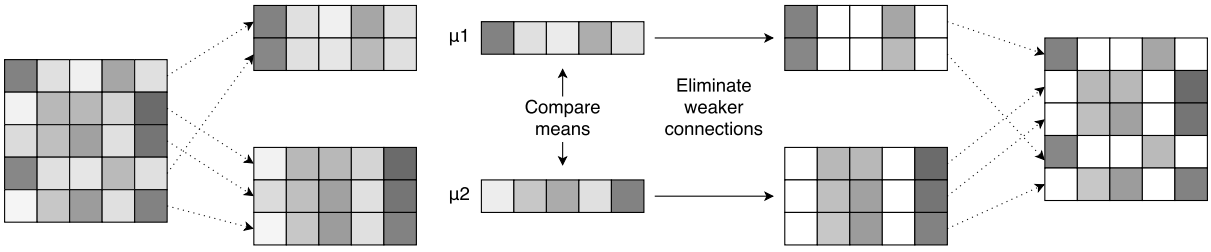


Figure 4: Grouping connections.

For the implementation, only the diagonal elements of the covariance matrix $\boldsymbol{\Sigma}_k$ were used with small noise added to make sure that its inverse matrix exists so that the normal distribution can be calculated. For the block matrix, a weight mask, $\tilde{W}$ which is of the same shape as W and has binary values that represent whether an element to be erased or remained, is created and multiplied to the matrix W by element-wise multiplication (Figure 12).

4.2 Optimize parameters

In our model, there exist two sets of parameters to optimize. One is the network parameters which are the weight matrices and bias vectors and the other is the Q parameters for the GMM. Let's denote them as $\boldsymbol{\theta}_{NN} = \{\mathbf{W}, \mathbf{b}\}$ and $\boldsymbol{\theta}_Q = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}\}$. To optimize these two sets of parameters at the same time, we introduce the following two functions.

First, consider an error function ERR_{NN}, which is the root mean squared error (RMSE) of the network output $\mathbf{y}$ compared to the target output $\mathbf{t}$.

$$ERR_{NN} = \sum_{i=1}^{p} (y_i - t_i)^2 \tag{4.2.1}$$

The parameters $\boldsymbol{\theta}_{NN}$ are trained to minimized the ERR_{NN} function.

Also, consider the $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$ function introduced in the Section 2.4.1. Let's denote the negative of the $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$ function as $NLL_Q = -Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$ which represents the negative log likelihood function. The parameters $\boldsymbol{\theta}_Q$ are trained to minimize the NLL_Q function.

To achieve both goals, we defined the cost function as $ERR_{NN} + NLL_Q$ and solve the optimization problem

$$\min_{\boldsymbol{\theta}_{NN}, \boldsymbol{\theta}_Q} Err_{NN} + \rho NLL_Q \tag{4.2.2}$$

Here, ρ, which is called the *Q learning rate* is the hyperparameter that controls the learning speed of $\boldsymbol{\theta}_Q$ separately from $\boldsymbol{\theta}_{NN}$. Both parameters are updated by gradient descent using backpropagation. As described in the Figure 5, errors back propagated from the NLL_Q term are stopped after updating the $\boldsymbol{\theta}_Q$ parameters to ensure that the parameters that are of the lower layers but not in $\boldsymbol{\theta}_Q$ are not affected by the NLL_Q term so that NLL_Q term only updates the parameters in $\boldsymbol{\theta}_Q$.

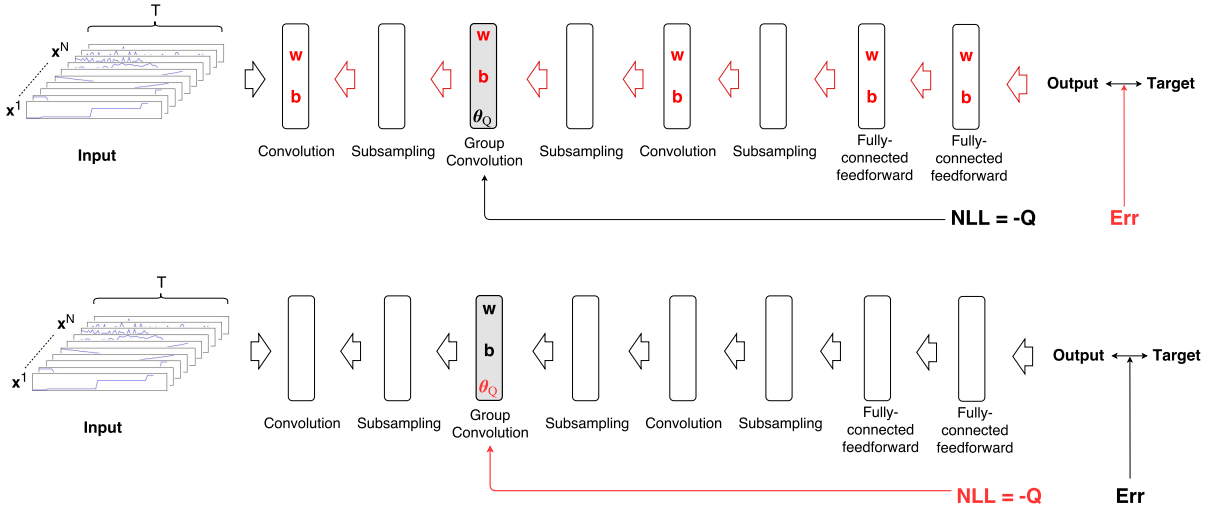


Figure 5: Optimization of a CNN whose second convolutional layer is grouped by the EM method. The cost function is combined with two terms, ERR_NN and NLL_Q and the error from each term updates two sets of parameters $\boldsymbol{\theta}_{NN}$ and $\boldsymbol{\theta}_Q$ separately.

The gradient updates for the $\boldsymbol{\theta}_Q$ from each observation $\mathbf{x}_i$ are as follow:

$$
\begin{aligned}
\frac{\partial}{\partial \mu_k} NLL_Q &= -\frac{\partial}{\partial \mu_k} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) \\
&= -\frac{\partial}{\partial \mu_k} \sum_k \gamma_{ik} \log p(\mathbf{x}_i|\boldsymbol{\theta}_k) \\
&= -\frac{\gamma_{ik}}{p(\mathbf{x}_i|\boldsymbol{\theta}_k)} \frac{\partial}{\partial \mu_k} p(\mathbf{x}_i|\boldsymbol{\theta}_k) \\
&= -\frac{1}{2} \frac{\gamma_{ik}}{\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \frac{(\boldsymbol{\Sigma}_k^{-1} + \boldsymbol{\Sigma}_k^{-T})(\mathbf{x}_i - \boldsymbol{\mu}_k)}{\sqrt{(2\pi)^L |\boldsymbol{\Sigma}_k|}} \exp(-0.5(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k)) \\
&= -\frac{\gamma_{ik}}{2} (\boldsymbol{\Sigma}_k^{-1} + \boldsymbol{\Sigma}_k^{-T})(\mathbf{x}_i - \boldsymbol{\mu}_k) \qquad (4.2.3) \\
\frac{\partial}{\partial \boldsymbol{\Sigma}_k} NLL_Q &= -\frac{\partial}{\partial \boldsymbol{\Sigma}_k} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) \\
&= -\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \sum_k \gamma_{ik} \log p(\mathbf{x}_i|\boldsymbol{\theta}_k) \\
&= -\frac{\gamma_{ik}}{\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \frac{\partial}{\partial \boldsymbol{\Sigma}_k} \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \\
&= -\frac{\gamma_{ik}}{\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \Big\{ \exp(F) \frac{\partial}{\partial \boldsymbol{\Sigma}_k} \frac{1}{\sqrt{(2\pi)^L |\boldsymbol{\Sigma}_k|}} + \frac{1}{\sqrt{(2\pi)^L |\boldsymbol{\Sigma}_k|}} \exp(F) \frac{\partial F}{\partial \boldsymbol{\Sigma}_k} \Big\} \\
&= -\frac{\gamma_{ik} \exp(F)}{\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \Big\{ \frac{\partial}{\partial \boldsymbol{\Sigma}_k} \frac{1}{\sqrt{(2\pi)^L |\boldsymbol{\Sigma}_k|}} + \frac{1}{\sqrt{(2\pi)^L |\boldsymbol{\Sigma}_k|}} \frac{\partial F}{\partial \boldsymbol{\Sigma}_k} \Big\} \\
&= -\frac{\gamma_{ik} \exp(F)}{\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \Big\{ -\frac{\text{tr}(\text{adj}(\boldsymbol{\Sigma}_k))}{2\sqrt{(2\pi)^L |\boldsymbol{\Sigma}_k|}^3} + \frac{\boldsymbol{\Sigma}_k^{-T} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-T}}{2\sqrt{(2\pi)^L |\boldsymbol{\Sigma}_k|}} \Big\} \\
&= \frac{\gamma_{ik}}{2} \Big\{ \frac{\text{tr}(\text{adj}(\boldsymbol{\Sigma}_k))}{(2\pi)^L |\boldsymbol{\Sigma}_k|} - \boldsymbol{\Sigma}_k^{-T} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-T} \Big\} \\
\frac{\partial}{\partial \pi_k} NLL_Q &= -\frac{\partial}{\partial \boldsymbol{\pi}_k} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) \\
&= -\frac{\partial}{\partial \boldsymbol{\pi}_k} \gamma_{ik} \log \boldsymbol{\pi}_k - \frac{\partial}{\partial \boldsymbol{\pi}_k} \gamma_{ik} \log p(\mathbf{x}_i|\boldsymbol{\theta}_k) \\
&= -\Big(\frac{\partial \gamma_{ik}}{\partial \boldsymbol{\pi}_k}\Big) \log \boldsymbol{\pi}_k - \frac{\gamma_{ik}}{\pi_k} - \Big(\frac{\partial \gamma_{ik}}{\partial \boldsymbol{\pi}_k}\Big) \log p(\mathbf{x}_i|\boldsymbol{\theta}_k) \qquad (4.2.4)
\end{aligned}
$$

In the derivation, F is substituted for the term inside the exponential function of the normal distribution, $-0.5(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k)$, and γ_{ik} and its derivative with respect to π_k are as below:

$$
\gamma_{ik} = \frac{\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k^{t-1})}{\sum_{k'} \pi_{k'} p(\mathbf{x}_i|\boldsymbol{\theta}_{k'}^{t-1})}, \qquad (4.2.5)
$$

$$
\frac{\partial \gamma_{ik}}{\partial \pi_k} = \frac{p(\mathbf{x}_i|\boldsymbol{\theta}^{t-1}) \sum_k' \pi_{k'} p(\mathbf{x}_i|\boldsymbol{\theta}^{t-1})}{\{\sum_{k'} \pi_{k'} p(\mathbf{x}|\boldsymbol{\theta}^{t-1})\}^2} \qquad (4.2.6)
$$

The derivatives with respect to each parameter are monotonically non-decreasing assuring that the functions are concave and can converge to some values.

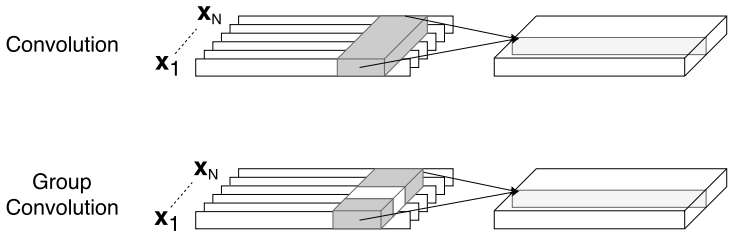


Figure 6: EM method applied time series convolution. Upper figure represents a general convolution for time series data $\mathbf{x}_1, ..., \mathbf{x}_N$. This convolution is same as 3D convolutions of image processing such that the height equals 1. Lower figure is the convolution applied with the EM method. One row of the kernel matrix is used to create one output channel.

4.3 Applying to CNNs

To process time series, we built CNNs with 3-dimensional convolutional layers in which one of the dimension is set to be 1 as in the Figure 6. In the figure, height of the layer is 1 so that the width represents the length of the sampled time series and each channel represents one time-series variable. In a convolutional layer of a CNN, channels of the input layer are multiplied by the kernels and the results are added to create one output channel (Figure 6). To apply EM method to a CNN, consider this convolution operation as a matrix multiplication such that the elements of the matrix are convolution kernels.

In Section 4.1, $\mathbf{w}_i$ were replaced for the $\mathbf{x}_i$ in the Section 2.4. However, in this case, $\mathbf{w}_{ij}$, the elements of $\mathbf{w}_i$ are also vectors. Concatenating the kernels into one long vector can be one method but instead, we feed the time series variables $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and perform the same process as in the Section 4.1.

This requires an assumption that if variables $\mathbf{x}_a$ and $\mathbf{x}_b$ have similar distribution, $\mathbf{w}_a$ and $\mathbf{w}_b$ also have similar distribution. In other words, the rows $\mathbf{w}_i$ should be in the same cluster when we perform clustering on the $\mathbf{x}_i$ if the weights connected to them are in the same cluster. Consider that two input variables have distributions that are similar enough. Then if those two variables are exchanged, the model would work the same. Therefore, we can assume that their weights would be also in the similar distributions.

Chapter V

Experiments

In experiments, we tested our method on the MNIST handwritten digit dataset to compare the model performance between a CNN which contains one or more group convolutional layer(s) and a CNN which does not.

5.1 Experimental Setup

MNIST dataset [1] is a large dataset of black and white hand written digit images. It is composed of 55000 train images, 5000 validation images, and 10000 test images and each image is composed of 28x28 pixels. Each image of 28x28 size was flattened to 1x784 vectors and fed to the model in the Figure 7.

The model is composed of three alternate layers of convolution and subsampling layers, and two fully connected layers. The first convolution layer enriches the input vector of 1 channel using convolutions of multiple kernels. The second convolution layer is replaced with group convolution which is applied the EM method to be compared with general CNN model. After three convolution and subsampling layers, it is flattened into a 1 dimensional vector and processed by two fully connected layers. The output layer has 10 nodes where each node represents each digit from 0 to 9. Linear function was used for the activation function and the softmax cross-entropy function is used for the ERR_{NN} function instead of the mean squared error function in the Section 4.2.

General CNN model and the model with a group convolution layer whose K values are set to 2, 3, and 4 are compared each other. The number of kernels for each model increase from CNN to CNN-2, CNN-3, and CNN-4 so the number of parameters used are similar. Detailed structures are in the Table 1.

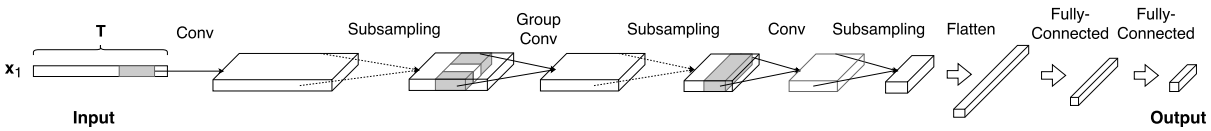


Figure 7: Structure of the model used for the experiments.

	Model	Input	Conv 1	Conv 2*	Conv 3	Flatten	Fully	Output
MNIST[1]	CNN	1x784x1	1x112x112	1x28x112	1x7x112	784	400	10
	CNN-2[2]	1x784x1	1x112x162	1x28x162	1x7x112	784	400	10
	CNN-3[3]	1x784x1	1x112x192	1x28x192	1x7x112	784	400	10
	CNN-4[4]	1x784x1	1x112x256	1x28x256	1x7x112	784	400	10

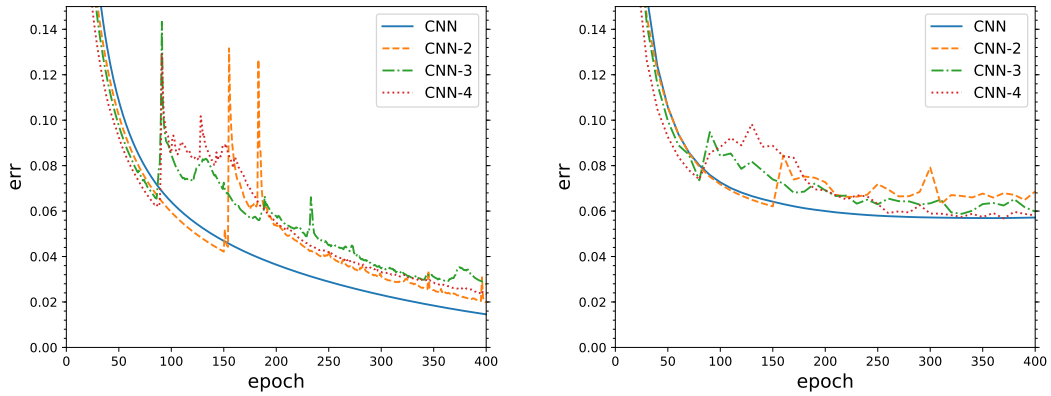
[1]MNIST Dataset. [2]K=2. [3]K=3. [4]K=4. *Output layer of a group convolution.

Table 1: Structure of the model used for the experiments. Conv 1, Conv 2, and Conv 3 of the model include additional subsampling layer after the convolution layer.

[1]http://yann.lecun.com/exdb/mnist/

5.2 Results

The models were trained for 400 epochs and validated every 10 epochs. The ERR_{NN} values are in the Figure 8. At the early steps of the training, the error values of the CNN-2, CNN-3, and NN-4 decrease faster than the baseline CNN model. However, after sudden increases of error values, the error values become bigger than the baseline model. Figure 9 shows ERR_{NN} and $Q = -NLL_Q$ changes on one figure. The sudden spikes of the error values overlap with the steep increases of Q values.



(a) Train error graph.

(b) Validation error graph.

Figure 8: Train and validation error graph. The horizontal axis is the training epoch and the vertical axis, err, represents the RMSE of the model output and the target output, which is the ERR_{NN} term of the Equation 4.2.1.

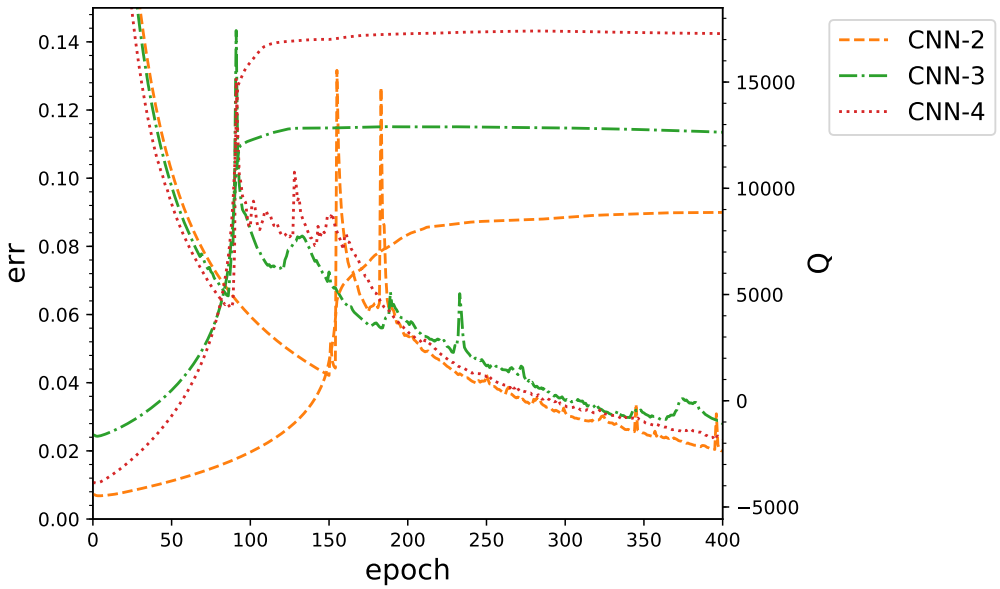
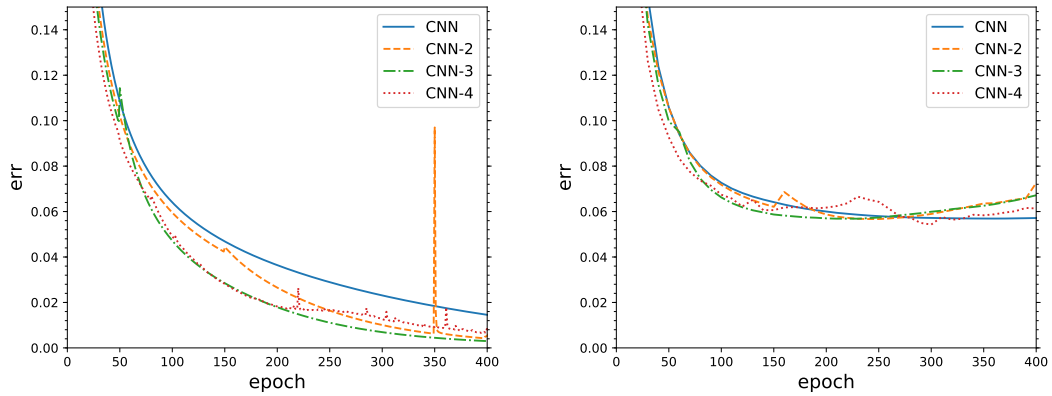


Figure 9: Train error and train Q graph. Left vertical axis is the ERR_{NN} and the right y-axis is the Q. The decreasing lines are the RMSE values that are on the left axis and the increasing lines are the Q values that are on the right axis.



(a) Train error graph. (b) Validation error graph.

Figure 10: Train and validation error graph. Only the ERR_{NN} of the Equation 4.2.1 is displayed. The gradient from NLL_Q is blocked after 150 epochs of training for the CNN-2, 80 epochs for the CNN-3, and 80 epochs for the CNN-4.

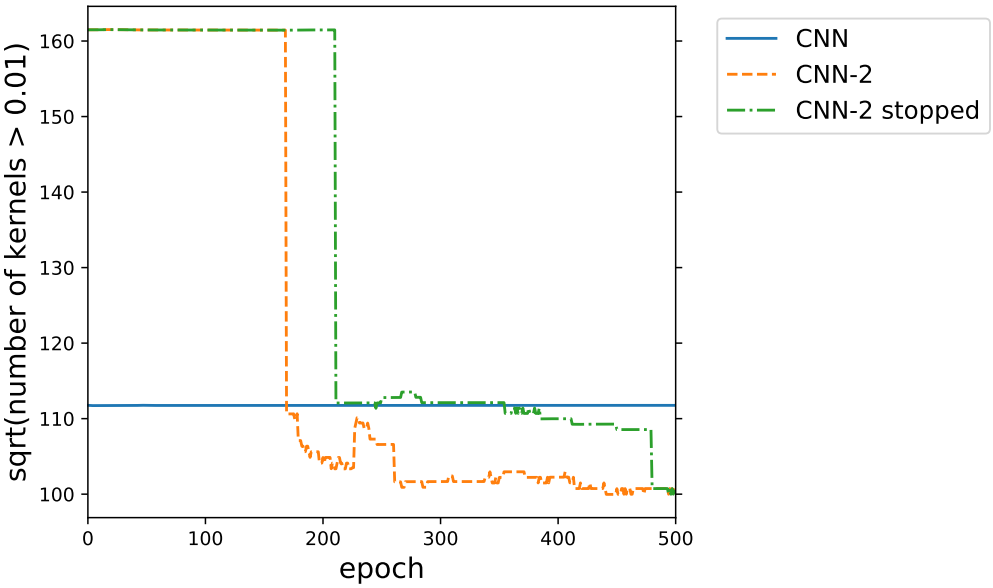
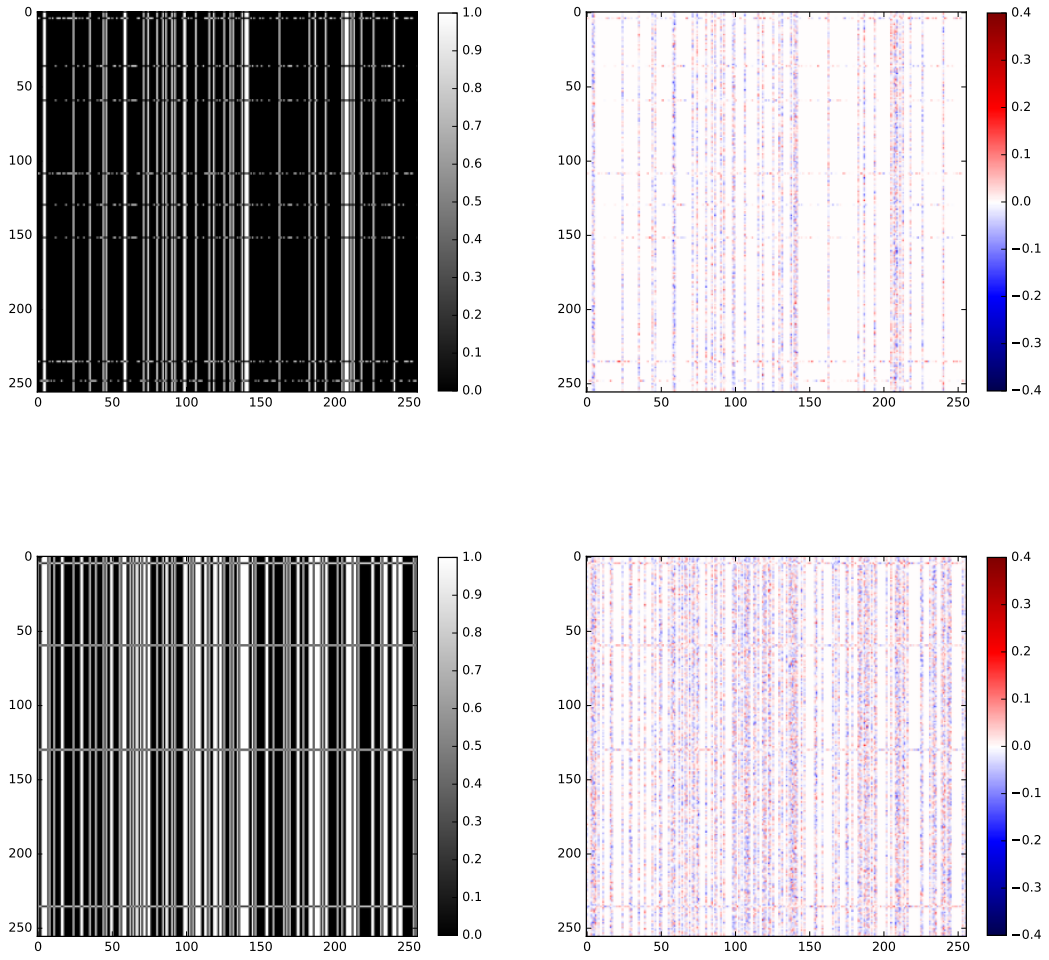


Figure 11: Change of the number of kernels whose mean of absolute value is greater than 0.01. *CNN-2 stopped* has the same shape as CNN-2 and the gradient from NLL_Q is stopped after 150 epochs of training. The vertical axis represents the square root of the actual digits.

To prevent this, the gradients from the NLL_Q term is blocked after certain epochs of training. Figure 10 shows the results. CNN-2, CNN-3, and CNN-4 models have lower train error than the CNN model. CNN with group convolution learn faster than the CNN model as their train errors decreases faster and the validation errors start to increase earlier than the CNN model.

CNN and CNN-2 models were trained again until 500 epochs to see the change of the parameters. Figure 11 shows the change of the number of kernels whose mean of absolute values are greater than 0.01. At early stages of training, CNN has 112×112 kernels while CNN-2 has 162×162 kernels. After



weight mask of CNN-4 masked weight of CNN-4

Figure 12: Experimental examples of the group structure in the Figure 1. Model CNN-4 of Table 1. Figures on the first row are from the model whose loss graphs are on the Figure 8 and the figures on the second row are from the model in which the gradient from the NLL_Q term is stopped after 80 epochs of training.

the model learns the group structure and prune the weights, CNN-2 uses even less kernels than CNN. Whether the gradient is stopped or not, the number of parameters decreases to the similar level and the network learns the group structure as in the Figure 12. Seeing that the number of kernels decreases later when the gradient is stopped, we can guess that the pruning occurred too early when the gradient is not blocked.

Chapter VI

Conclusion and Future Work

Multivariate time series analysis is applicable to a wide range of real world problems. Machine learning methods are used to deal with the problems and CNNs can be used to extract features from the multivariate sequential data. However, CNNs suffer from the redundant kernels like other ANNs. An EM method can learn the group structure of the convolution layers by end-to-end backpropagation learning. The group structure can be exploited by reducing the redundant kernels and adding more emphasis on the strong connections. In the experiments with MNIST dataset, the CNNs with a group convolution layer were trained faster than the baseline CNN model by building more connections at the early stage of learning and pruning unnecessary connections based on the trained group structure.

The experiment showed that the deep neural networks can be expressed using less connections if we use the group structure. EM method was applied to only the CNN for the time series dataset but for the future work it can be applied to other types of ANNs such as multilayer perceptrons (MLPs). Also, the group convolution calculation need to be optimized as it takes too much time than normal convolution due to the probability distribution calculation of GMM.

References

[1] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[2] Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4):291–294.

[3] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceeding of the International Conference on Machine learning*, pages 160–167. ACM.

[4] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

[5] Graves, A., Mohamed, A., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

[6] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[7] Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv:1404.2188*.

[8] Krause, A., Singh, A., and Guestrin, C. (2008). Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Machine Learning Research*, 9(Feb):235–284.

[9] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1097–1105.

[10] LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time-series. *The handbook of brain theory and neural networks*, 3361(10):255–258.

[11] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324.

[12] Liang, M. and Hu, X. (2015). Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3367–3375.

[13] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440.

[14] Mo, Y. and Sinopoli, B. (2015). Secure estimation in the presence of integrity attacks. *IEEE Transactions on Automatic Control*, 60(4):1145–1151.

[15] Murphy, K. P. (2012). *Adaptive Computation and Machine Learning : Machine Learning : A Probabilistic Perspective*. MIT Press.

[16] Ordóñez, F. J. and Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115.

[17] Rasmussen, C. E. (2006). Gaussian processes for machine learning. MIT Press.

[18] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proceedings of Advances in neural information processing systems*, pages 91–99.

[19] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

[20] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

[21] Yang, J. B., Nguyen, M. N., San, P. P., Li, X. L., and Krishnaswamy, S. (2015). Deep convolutional neural networks on multichannel time series for human activity recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 25–31.

[22] Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67.

[23] Zemel, R. S. (1994). Autoencoders, minimum description length and helmholtz free energy. Proceedings of the Neural Information Processing Systems Conference.

[24] Zheng, Y., Liu, Q., Chen, E., Ge, Y., and Zhao, J. *Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks*, pages 298–310. Springer International Publishing.

Acknowledgement

This thesis could be complete with the support and help of many people. I would like to give my gratitude to Pf. Jaesik Choi for all the advises and teaching he has given to me. Also, I'd like to thank to all the members of the Statistical Artificial Intelligence Lab who spent a lot of time together in the same place. I learned lots of things from the discussions we had. I would like to thank to my friends who was there for me when I needed and supported me to finish my research. Lastly, I would like to express my gratitude to my parents and brother. I would have not been able to finish my work without them.